

INTERFAZ GRÁFICO DE USUARIO

7.1 CREACIÓN DE VENTANAS

7.1.1 *Introducción*

Hasta ahora, todos los ejemplos que se han desarrollado se han ejecutado en modo consola, es decir, las entradas de datos y la visualización de resultados se realiza a través de una ventana de MS-DOS. Las aplicaciones normalmente requieren de un interfaz de usuario más sofisticado, basado en ventanas gráficas y distintos componentes (botones, listas, cajas de texto, etc.) en el interior de estas ventanas.

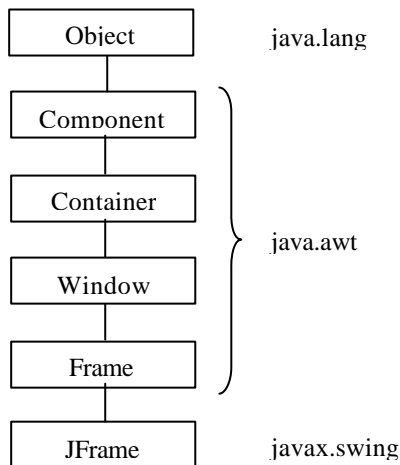
Java proporciona una serie de paquetes estándar que nos facilitan la labor de crear los Interfaces Gráficos de Usuario (GUI) que necesitamos. Estos paquetes se encuentran englobados dentro de Java Foundation Classes (JFC), que proporciona:

- Soporte de componentes de entrada/salida a través de AWT (Abstract Window Toolkit) y Swing
- Imágenes 2D
- Servicio de impresión
- Soporte para funciones Drag-and-Drop (arrastrar y pegar)
- Funciones de accesibilidad, etc.

De los paquetes anteriores, los más utilizados, con diferencia, son AWT y Swing. AWT fue el primero en aparecer y proporciona los componentes básicos de entrada/salida; Swing recoge la mayor parte de la funcionalidad de AWT, aportando clases más complejas y especializadas que las ya existentes, así como otras de nueva creación.

El elemento más básico que se suele emplear en un interfaz gráfico de usuario es la ventana. AWT proporciona la clase *Window*, aunque para simplificar los desarrollos acudiremos a la clase *Frame* (marco) derivada de *Window* y por lo tanto más especializada.

La jerarquía existente en torno a las ventanas de Java es:



La clase *Object* es la raíz de toda la jerarquía de clases, por lo que todos los objetos tienen a *Object* como superclase. La clase abstracta *Component* proporciona una representación gráfica susceptible de ser representada en un dispositivo de salida; como se puede observar es una clase muy genérica que no utilizaremos directamente, sino a través de sus clases derivadas.

La clase abstracta *Container* deriva de *Component* y añade la funcionalidad de poder contener otros componentes AWT (por ejemplo varios botones, una caja de texto, etc.). La clase *Window* proporciona una ventana gráfica sin bordes ni posibilidad de incluir barras de menú, por ello haremos uso de su clase derivada *Frame*, que no presenta las restricciones del objeto *Window*.

JFrame es la versión de *Frame* proporcionada por *Swing*. Al igual que muchos otros componentes de *Swing*, ofrece posibilidades más amplias que las implementadas por sus superclases de AWT, aunque también presenta una mayor complejidad de uso.

Nosotros haremos uso de las clases más útiles que nos ofrece AWT, empezando por la definición de ventanas mediante el objeto *Frame*.

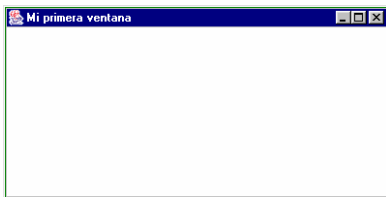
7.1.2 Utilización de la clase *Frame*

Una manera muy sencilla de crear y visualizar una ventana en Java se muestra en el siguiente ejemplo (*Marco1*): en primer lugar se importa la clase *Frame* de AWT (línea 1); más adelante, en otros ejemplos, utilizaremos conjuntamente diferentes clases de AWT, por lo que nos resultará más sencillo utilizar la instrucción *import java.awt.**.

La línea 6 nos crea la instancia *MiMarco* de tipo *Frame*; en la línea 7 se asigna un tamaño de ventana de 400 (horizontal) por 200 (vertical) pixels; en la línea 8 se define el título de la ventana y, por último, la línea 9 hace visible (representa) el marco creado y definido en las líneas anteriores.

La clase *Frame* contiene 4 constructores; uno de ellos permite definir el título, por lo que las líneas 6 y 8 pueden sustituirse por la sentencia *Frame MiMarco = new Frame("Mi primera ventana");*.

```
1  import java.awt.Frame;
2
3  public class Marco1 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          MiMarco.setSize(400,200);
8          MiMarco.setTitle("Mi primera ventana");
9          MiMarco.setVisible(true);
10     }
11 }
```



No existe ningún constructor de la clase *Frame* que admita como parámetros el tamaño de la ventana, lo que nos ofrece la posibilidad de crearnos nuestra propia clase (*Marco2*) que herede de *Frame* y ofrezca esta posibilidad:

```
1  import java.awt.Frame;
2
3  public class Marco2 extends Frame {
4
5      Marco2(String Titulo) {
```

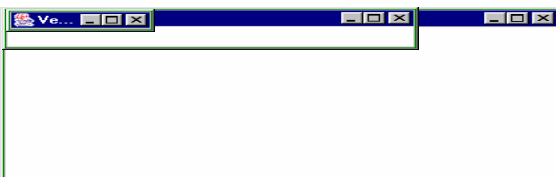
```
6      this.setTitle(Titulo);
7      this.setVisible(true);
8  }
9
10     Marco2(String Titulo, int Ancho, int Alto) {
11         this(Titulo);
12         this.setSize(Ancho, Alto);
13     }
14
15     Marco2() {
16         this("Mi primera ventana",400,200);
17     }
18
19 }
```

La clase *Marco2* que hemos diseñado permite utilizar el constructor vacío (línea 15), obteniendo una ventana de tamaño 400 x 200 con el título “Mi primera ventana”. También se puede incorporar el título (línea 5), consiguiéndose una ventana visible (línea 7) con el título indicado. Finalmente se proporciona un constructor que permite incluir título y dimensiones (línea 10); en este caso nos apoyamos en el constructor que admite título (palabra reservada *this* en la línea 11) y el método *setSize* del objeto *Frame* (línea 12).

Una vez definida la clase *Marco2*, podemos utilizarla en diferentes programas como el mostrado a continuación:

```
1  public class Marco2main {
2
3      public static void main(String[] args) {
4          Marco2 MiVentana1 = new Marco2();
5          Marco2 MiVentana2 = new Marco2("Ventana2",300,50);
6          Marco2 MiVentana3 = new Marco2("Ventana3");
7      }
8  }
```

Marco2main crea tres instancias de la clase *Marco2*, que al ser subclase de *Frame* representa marcos de ventanas; las líneas 4 a 6 muestran diferentes maneras de instanciar *Marco2*. El resultado obtenido son tres ventanas superpuestas de diferentes tamaños y títulos:



7.1.3 Creación y posicionamiento de múltiples ventanas

Los interfaces gráficos de usuario (GUI) no tienen por qué restringirse al uso de una única ventana para entrada de datos, mostrar información, etc. Siguiendo el mecanismo explicado en los apartados anteriores podemos crear y hacer uso de un número ilimitado de ventanas.

Cuando se crean varias ventanas a menudo resulta útil situar cada una de ellas en diferentes posiciones del dispositivo de salida, de forma que adopten una configuración práctica y estética, evitándose que se oculten entre sí, etc.

Para situar una ventana en una posición concreta utilizamos el método *setLocation*, heredado de la clase *Component*. Este método está sobrecargado y admite dos tipos de parámetros de entrada:

- *setLocation* (int x, int y)
- *setLocation* (Point p)

En ambos casos es necesario suministrar las coordenadas bidimensionales que sitúen la esquina superior izquierda de la ventana respecto a su componente padre (la pantalla, el objeto contenedor del *Frame*, etc.). El siguiente ejemplo (*Marco3*) ilustra la manera de utilizar el método que usa la clase *Point* de AWT como argumento:

```
1  import java.awt.Frame;
2  import java.awt.Point;
3
4  public class Marco3 {
5
6      public static void main(String[] args) {
7          Frame MiMarco = new Frame();
8          MiMarco.setSize(400,200);
9          MiMarco.setTitle("Mi primera ventana");
10         MiMarco.setLocation(new Point(100,220));
11         MiMarco.setVisible(true);
12     }
13 }
```

El resultado de la clase *Marco3* es una ventana situada en la posición (100,220) respecto al origen del *GraphicsDevice*, que habitualmente es la pantalla gráfica del ordenador.

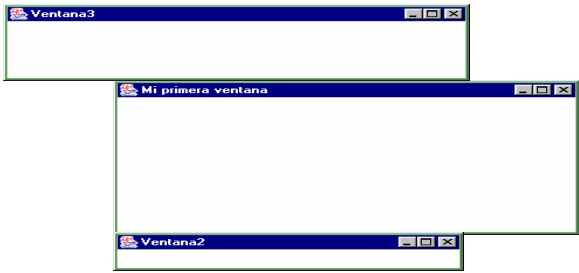
A continuación se presenta la clase *Marco4*, que surge de *Marco2*, añadiendo la posibilidad de indicar la posición de la ventana instanciada:

```
1  import java.awt.Frame;
2  import java.awt.Point;
3
4  public class Marco4 extends Frame {
5
6      Marco4(String Titulo) {
7          this.setTitle(Titulo);
8          this.setVisible(true);
9      }
10
11     Marco4(String Titulo, int Ancho, int Alto) {
12         this(Titulo);
13         this.setSize(Ancho, Alto);
14     }
15
16     Marco4(String Titulo, int Ancho, int Alto,
17             int PosX, int PosY) {
18         this(Titulo,Ancho, Alto);
19         this.setLocation(new Point(PosX,PosY));
20     }
21
22     Marco4() {
23         this("Mi primera ventana",400,200,100,100);
24     }
25
26 }
```

El constructor definido en la línea 16 permite situar el marco creado en la posición deseada (línea 19). La clase *Marco4main* instancia varias ventanas con diferentes tamaños y posiciones iniciales:

```
1  public class Marco4main {
2
3      public static void main(String[] args) {
4          Marco4 MiVentana1 = new Marco4();
5          Marco4 MiVentana2 = new
6                                  Marco4("Ventana2",300,50,100,300);
7          Marco4 MiVentana3 = new Marco4("Ventana3",400,100);
8      }
```

La posición relativa de las 3 ventanas involucradas en el ejemplo es la siguiente:



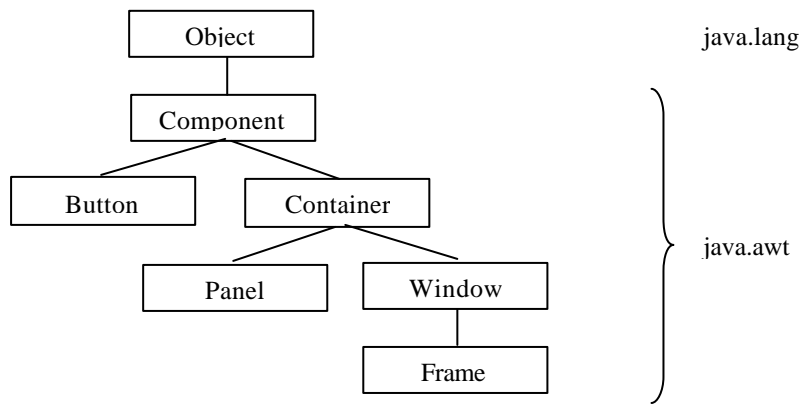
7.2 PANELES Y OBJETOS DE DISPOSICIÓN (LAYOUTS)

7.2.1 Introducción

Las ventanas tienen utilidad si podemos añadirles diversos componentes de interacción con los usuarios. El elemento que en principio se adapta mejor a esta necesidad es la clase *Container*; como ya se explicó: “La clase abstracta *Container* deriva de *Component* y añade la funcionalidad de poder contener otros componentes AWT (por ejemplo varios botones, una caja de texto, etc.)”.

En lugar de utilizar directamente la clase *Container*, resulta mucho más sencillo hacer uso de su clase heredada *Panel*. Un panel proporciona espacio donde una aplicación puede incluir cualquier componente, incluyendo otros paneles.

Para ilustrar el funcionamiento y necesidad de los paneles emplearemos un componente muy útil, conocido y fácil de utilizar: los botones (*Button*). Una forma de crear un botón es a través de su constructor *Button(String Texto)*, que instancia un botón con el “Texto” interior que se le pasa como parámetro.



Nuestro primer ejemplo (*Boton1*) no hace uso de ningún panel, situando un botón directamente sobre una ventana. En la línea 8 se crea la instancia *BotonHola* del objeto *Button*, con el texto “Hola” como etiqueta. La línea 10 es muy importante, añade el componente *BotonHola* al contenedor *MiMarco*; además de crear los componentes (en nuestro caso el botón) es necesario situar los mismos en los contenedores donde se vayan a visualizar.

A continuación del código se representa el resultado del programa: un botón que ocupa toda la ventana. Si hubiéramos creado varios botones y los hubiésemos añadido a *MiMarco*, el resultado visible sería la ventana ocupada por entero por el último botón añadido.

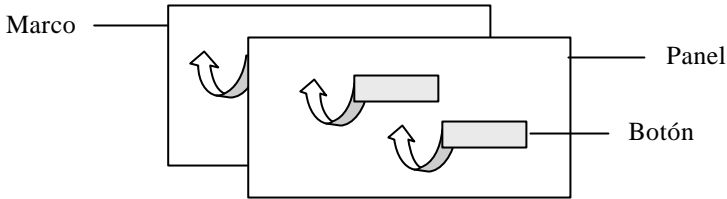
```
1  import java.awt.Frame;
2  import java.awt.Button;
3
4  public class Boton1 {
5
6      public static void main(String[] args) {
7          Frame MiMarco = new Frame();
8          Button BotonHola = new Button("Hola");
9
10         MiMarco.add(BotonHola);
11
12         MiMarco.setSize(400,200);
13         MiMarco.setTitle("Ventana con botón");
14         MiMarco.setVisible(true);
15     }
16 }
```



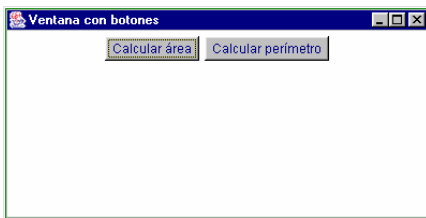
Si no utilizamos paneles (y/o *layouts*), los componentes (tales como botones) se representarán en las ventanas de una manera inadecuada. Los paneles y *layouts* nos permiten añadir componentes visuales con diferentes disposiciones en el plano.

El ejemplo anterior puede ser ampliado con facilidad para que el marco incorpore un panel. Usando el panel conseguiremos que los componentes que añadamos al mismo sigan una disposición establecida (por defecto ordenación secuencial).

La clase *Boton2* instancia un panel en la línea 9 y lo añade al marco en la línea 13. Los botones creados en las líneas 10 y 11 se añaden (secuencialmente, por defecto) al panel en las líneas 14 y 15:

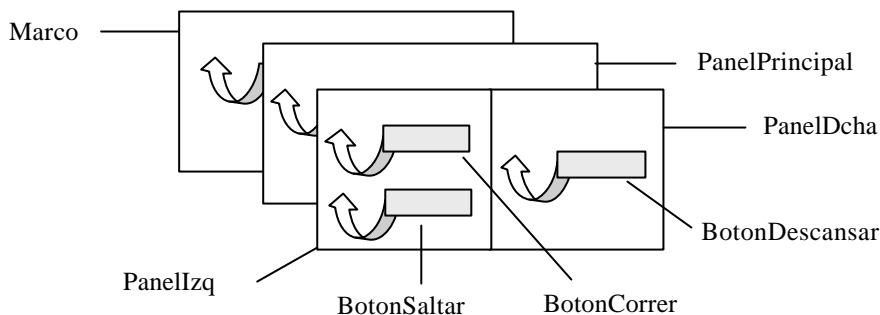


```
1  import java.awt.Frame;
2  import java.awt.Button;
3  import java.awt.Panel;
4
5  public class Boton2 {
6
7      public static void main(String[] args) {
8          Frame MiMarco = new Frame();
9          Panel MiPanel = new Panel();
10         Button BotonArea = new Button("Calcular área");
11         Button BotonPerimetro = new Button("Calcular
                                         perímetro");
12
13         MiMarco.add(MiPanel);
14         MiPanel.add(BotonArea);
15         MiPanel.add(BotonPerimetro);
16
17         MiMarco.setSize(400,200);
18         MiMarco.setTitle("Ventana con botones");
19         MiMarco.setVisible(true);
20     }
21 }
```



7.2.2 Utilización básica de paneles

Los paneles son contenedores que pueden albergar componentes (incluidos otros paneles). En el siguiente ejemplo (*Panel1*) emplearemos la disposición de elementos que se muestra a continuación:



```

1  import java.awt.Frame;
2  import java.awt.Button;
3  import java.awt.Panel;
4
5  public class Panel1 {
6
7      public static void main(String[] args) {
8          Frame MiMarco = new Frame();
9          Panel PanelPrincipal = new Panel();
10         Panel PanelIzq = new Panel();
11         Panel PanelDcha = new Panel();
12
13         Button BotonCorrer = new Button("Correr");
14         Button BotonSaltar = new Button("Saltar");
15         Button BotonDescansar = new Button("Descansar");
16
17         MiMarco.add(PanelPrincipal);
18         PanelPrincipal.add(PanelIzq);
19         PanelPrincipal.add(PanelDcha);
20         PanelIzq.add(BotonCorrer);
21         PanelIzq.add(BotonSaltar);
22         PanelDcha.add(BotonDescansar);
23
24         MiMarco.setSize(400,200);
25         MiMarco.setTitle("Ventana con paneles");
26         MiMarco.setVisible(true);
27     }
28 }

```

La ventana de la izquierda representa un posible resultado de este programa. Dependiendo de la definición de pantalla en pixels que tenga el usuario la relación de tamaños entre los botones y la ventana puede variar. Si los botones no caben a lo ancho, la disposición secuencial hará “saltar de línea” al panel derecho, obteniéndose una visualización como la mostrada en la ventana de la derecha.



La idea subyacente en el diseño del paquete AWT es que los interfaces gráficos de usuario puedan visualizarse independientemente de las características y configuración de los dispositivos de salida, así como del tamaño de las ventanas de visualización (en muchos casos los navegadores Web). Para compaginar este objetivo con la necesidad de dotar al programador de la flexibilidad necesaria en el diseño de GUI's se proporcionan las clases “Layouts” (*FlowLayout*, *BorderLayout*, *GridLayout*, etc.).

Cada panel que creamos tiene asociado, implícita o explícitamente, un tipo de disposición (“layout”) con la que se visualizan los componentes que se le añaden. Si no especificamos nada, la disposición por defecto (implícita) es *FlowLayout*, con los elementos centrados en el panel, tal y como aparecen los botones del ejemplo anterior.

7.2.3 Objeto de disposición (Layout): *FlowLayout*

Para establecer una disposición de manera explícita, podemos recurrir al método *setLayout* (heredado de la clase *Container*) o bien instanciar nuestro panel indicando directamente el “layout” deseado. Las siguientes instrucciones ilustran las posibles maneras de asociar un *FlowLayout* a un panel:

```
Panel MiPanel = new Panel (new FlowLayout());
```

```
OtroPanel.setLayout(new FlowLayout());
```

El siguiente ejemplo (*FlowLayout1*) muestra una posible forma de crear una ventana con 4 botones dispuestos horizontalmente. En la línea 11 creamos el objeto

PosicionamientoSecuencial, de tipo *FlowLayout*. En la línea 17 asociamos este objeto al panel instanciado en la línea 10 (*MiPanel*).

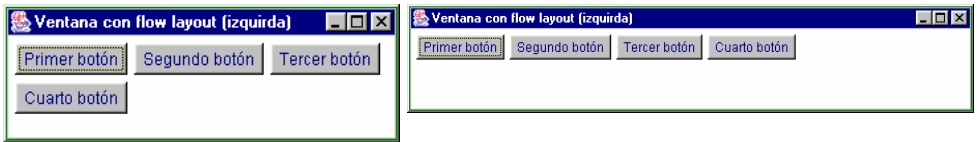
```
1  import java.awt.Frame;
2  import java.awt.Button;
3  import java.awt.Panel;
4  import java.awt.FlowLayout;
5
6  public class FlowLayout1 {
7
8      public static void main(String[] args) {
9          Frame MiMarco = new Frame();
10         Panel MiPanel = new Panel();
11         FlowLayout PosicionamientoSecuencial = new
                                                    FlowLayout();
12         Button BotonA = new Button("Primer botón");
13         Button BotonB = new Button("Segundo botón");
14         Button BotonC = new Button("Tercer botón");
15         Button BotonD = new Button("Cuarto botón");
16
17         MiPanel.setLayout(PosicionamientoSecuencial);
18
19         MiMarco.add(MiPanel);
20         MiPanel.add(BotonA);
21         MiPanel.add(BotonB);
22         MiPanel.add(BotonC);
23         MiPanel.add(BotonD);
24
25         MiMarco.setSize(300,100);
26         MiMarco.setTitle("Ventana con flow layout");
27         MiMarco.setVisible(true);
28     }
29 }
```

La ventana que se visualiza contiene cuatro botones posicionados consecutivamente y centrados respecto al panel. Dependiendo del tamaño con el que dimensionemos la ventana, los botones cabrán horizontalmente o bien tendrán que “saltar” a sucesivas líneas para tener cabida, tal y como ocurre con las líneas en un procesador de texto:



Los objetos *FlowLayout* pueden instanciarse con posicionamiento centrado (*FlowLayout.CENTER*), izquierdo (*FlowLayout.LEFT*), derecho (*FlowLayout.RIGHT*) y los menos utilizados *LEADING* y *TRAILING*. La disposición por defecto es *CENTER*.

Si en el ejemplo anterior sustituimos la línea 11 por: *FlowLayout PosicionamientoSecuencial = new FlowLayout(FlowLayout.LEFT);* el resultado obtenido es:



7.2.4 Objeto de disposición (Layout): BorderLayout

Un *BorderLayout* sitúa y dimensiona los componentes en 5 “regiones cardinales”: norte, sur, este, oeste y centro. La clase que se presenta a continuación (*BorderLayout1*) crea un *BorderLayout* (línea 11) y lo asocia al panel *MiPanel* (línea 18). En las líneas 21 a 25 se añaden cinco botones al panel, cada uno en una zona “cardinal”.

```
1 import java.awt.Frame;
2 import java.awt.Button;
3 import java.awt.Panel;
4 import java.awt.BorderLayout;
5
6 public class BorderLayout1 {
7
8     public static void main(String[] args) {
9         Frame MiMarco = new Frame();
10        Panel MiPanel = new Panel();
11        BorderLayout PuntosCardinales = new BorderLayout();
12        Button BotonNorte = new Button("Norte");
13        Button BotonSur = new Button("Sur");
14        Button BotonEste = new Button("Este");
15        Button BotonOeste = new Button("Oeste");
16        Button BotonCentro = new Button("Centro");
17
18        MiPanel.setLayout(PuntosCardinales);
19
```

```

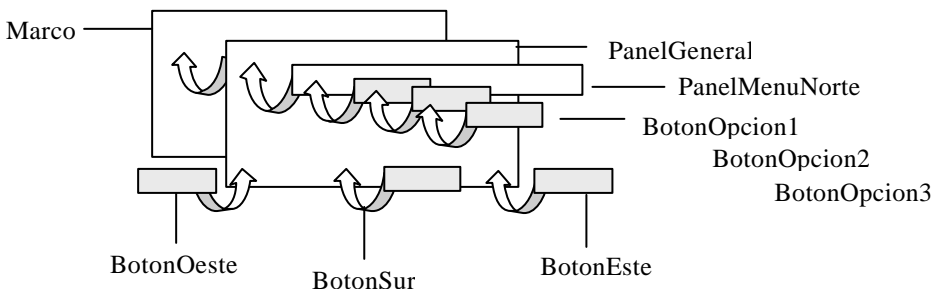
20     MiMarco.add(MiPanel);
21     MiPanel.add(BotonNorte, BorderLayout.NORTH);
22     MiPanel.add(BotonSur, BorderLayout.SOUTH);
23     MiPanel.add(BotonEste, BorderLayout.EAST);
24     MiPanel.add(BotonOeste, BorderLayout.WEST);
25     MiPanel.add(BotonCentro, BorderLayout.CENTER);
26
27     MiMarco.setSize(300,100);
28     MiMarco.setTitle("Ventana con BorderLayout");
29     MiMarco.setVisible(true);
30 }
31 }

```

Tal y como muestran los dos siguientes resultados (correspondientes a la ventana con el tamaño original y la misma ventana agrandada) los componentes (botones) insertados en el panel se sitúan en sus respectivas regiones del *BorderLayout*.



Cada región de un *BorderLayout* admite un único componente, pero nada impide que ese componente sea un contenedor que admita una nueva serie de componentes. Vamos a modificar la clase *BorderLayout1* para que la región norte sea ocupada por tres botones dispuestos en posición horizontal. El diseño de la solución es el siguiente:



En la clase *BorderLayout2* se crea el panel *PanelGeneral* (línea 8) al que se asocia el *BorderLayout PuntosCardinales* (línea 9) mediante el método *setLayout* (línea 19). Análogamente, se crea el panel *PanelMenuNorte* (línea 7) al que se asocia el *FlowLayout OpcionesMenu* (línea 10) mediante el método *setLayout* (línea 20).

A *PanelMenuNorte* se le añaden tres botones (líneas 24 a 26), mientras que el propio panel (con los tres botones incluidos) se añade a la región norte del *PanelGeneral* (línea 23). De esta manera, el botón *BotonNorte* del ejemplo anterior queda sustituido por la secuencia (en flujo secuencial) de botones *BotonOpcion1*, *BotonOpcion2* y *BotonOpcion3*.

```
1  import java.awt.*;
2
3  public class BorderLayout2 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel PanelMenuNorte = new Panel();
8          Panel PanelGeneral = new Panel();
9          BorderLayout PuntosCardinales = new BorderLayout();
10         FlowLayout OpcionesMenu = new FlowLayout();
11         Button BotonOpcion1 = new Button("Opción 1");
12         Button BotonOpcion2 = new Button("Opción 2");
13         Button BotonOpcion3 = new Button("Opción 3");
14         Button BotonSur = new Button("Sur");
15         Button BotonEste = new Button("Este");
16         Button BotonOeste = new Button("Oeste");
17         Button BotonCentro = new Button("Centro");
18
19         PanelGeneral.setLayout(PuntosCardinales);
20         PanelMenuNorte.setLayout(OpcionesMenu);
21
22         MiMarco.add(PanelGeneral);
23         PanelGeneral.add(PanelMenuNorte, BorderLayout.NORTH);
24         PanelMenuNorte.add(BotonOpcion1);
25         PanelMenuNorte.add(BotonOpcion2);
26         PanelMenuNorte.add(BotonOpcion3);
27         PanelGeneral.add(BotonSur, BorderLayout.SOUTH);
28         PanelGeneral.add(BotonEste, BorderLayout.EAST);
29         PanelGeneral.add(BotonOeste, BorderLayout.WEST);
30         PanelGeneral.add(BotonCentro, BorderLayout.CENTER);
31
32         MiMarco.setSize(400,150);
33         MiMarco.setTitle("Ventana con BorderLayout");
34         MiMarco.setVisible(true);
35     }
36 }
```



7.2.5 Objeto de disposición (Layout): *GridLayout*

Un *GridLayout* coloca y dimensiona los componentes en una rejilla rectangular. El contenedor se divide en rectángulos de igual tamaño, colocándose un componente en cada rectángulo.

Los objetos *GridLayout* se pueden instanciar indicando el número de filas y columnas de la rejilla:

```
GridLayout Matriz = new GridLayout(3,2);
```

También se puede hacer uso del constructor vacío y posteriormente utilizar los métodos *setRows* y *setColumns*:

```
GridLayout Matriz = new GridLayout();
```

```
Matriz.setRows(3);
```

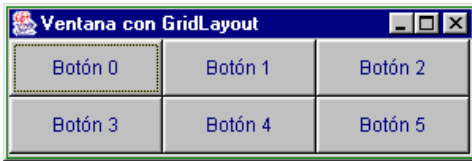
```
Matriz.setColumns(2);
```

La clase *GridLayout1* crea una ventana con 6 botones dispuestos matricialmente en 2 filas y 3 columnas, para ello se instancia un objeto de tipo *GridLayout* con dicha disposición (línea 9). En la línea 14 se asocia el *GridLayout* al panel, añadiéndose posteriormente 6 botones al mismo (líneas 15 y 16).

```
1 import java.awt.*;
2
3 public class GridLayout1 {
4
5     public static void main(String[] args) {
6         Frame MiMarco = new Frame();
7         Panel MiPanel = new Panel();
8
9         GridLayout Matriz = new GridLayout(2,3);
```

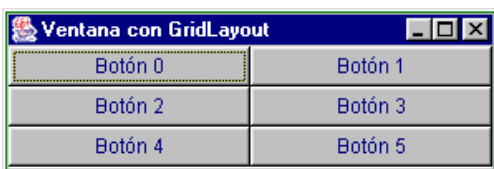


```
10     Button[] Botones = new Button[6];
11     for (int i=0;i<6;i++)
12         Botones[i] = new Button("Botón "+i);
13
14     MiPanel.setLayout(Matriz);
15     for (int i=0;i<6;i++)
16         MiPanel.add(Botones[i]);
17
18     MiMarco.add(MiPanel);
19     MiMarco.setSize(300,100);
20     MiMarco.setTitle("Ventana con GridLayout");
21     MiMarco.setVisible(true);
22 }
23 }
```



En *GridLayout2* se crea una clase muy sencilla en la que se asocia un *GridLayout* de 3 x 2 directamente al marco de la aplicación:

```
1  import java.awt.*;
2
3  public class GridLayout2 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7
8          MiMarco.setLayout(new GridLayout(3,2));
9          for (int i=0;i<6;i++)
10             MiMarco.add(new Button("Botón "+i));
11
12         MiMarco.setSize(300,100);
13         MiMarco.setTitle("Ventana con GridLayout");
14         MiMarco.setVisible(true);
15     }
16 }
```



Finalmente, en *GridLayout3* se utiliza un *GridLayout* (línea 8) en el que, en una celda, se añade un componente *Panel* (*Vertical*) con disposición *BorderLayout* (línea 7). Al panel *Vertical* se le añaden tres botones en las regiones cardinales: norte, centro y sur (líneas 10 a 12).

```
1  import java.awt.*;
2
3  public class GridLayout3 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel Vertical = new Panel(new BorderLayout());
8          MiMarco.setLayout(new GridLayout(2,3));
9
10         Vertical.add(new Button("Arriba"), BorderLayout.NORTH);
11         Vertical.add(new Button("Centro"),
12                     BorderLayout.CENTER);
13
14         Vertical.add(new Button("Abajo"), BorderLayout.SOUTH);
15
16         MiMarco.add(Vertical);
17         for (int i=1;i<6;i++)
18             MiMarco.add(new Button("Botón "+i));
19
20         MiMarco.setSize(300,160);
21         MiMarco.setTitle("Ventana con Layouts Grid y Border");
22         MiMarco.setVisible(true);
23     }
24 }
```

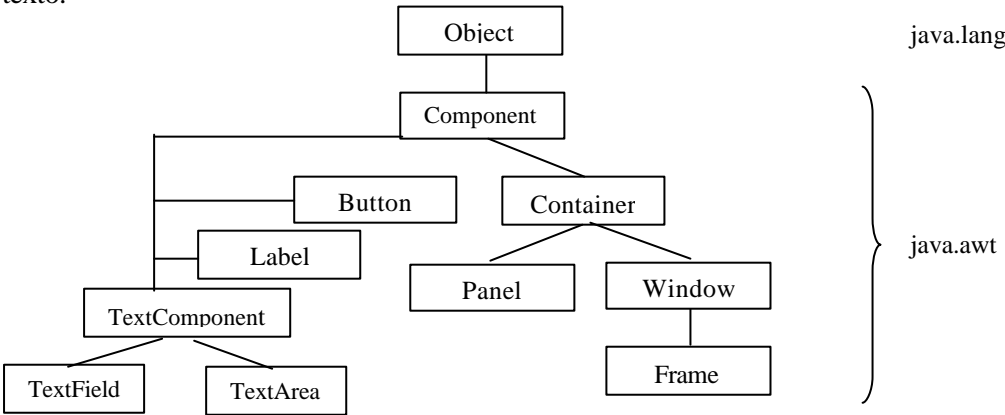


7.3 ETIQUETAS, CAMPOS Y ÁREAS DE TEXTO

7.3.1 Introducción

Una vez que somos capaces de crear marcos, añadirles paneles con diferentes “layouts” e insertar componentes en los mismos, estamos preparados para realizar diseños de interfaces de usuario, sin embargo, todavía nos falta conocer los diferentes componentes básicos de entrada/salida que podemos utilizar en Java.

Un componente que hemos empleado en todos los ejemplos anteriores es el botón (*Button*). Ahora se explicarán las etiquetas, los campos de texto y las áreas de texto.



7.3.2 Etiqueta (Label)

Las etiquetas permiten situar un texto en un contenedor. El usuario no puede editar el texto, aunque si se puede variar por programa.

Los constructores de la clase *Label* son:

- Label ()*
- Label (String Texto)*
- Label (String Texto, int Alineacion)*

Entre los métodos existentes tenemos:

setText (String Texto)

setAlignment (int Alineacion)

De esta manera podemos definir e instanciar etiquetas de diversas maneras:

Label Saludo = new Label ("Hola", Label.LEFT);

Label OtroSaludo = new Label ("Buenos días");

OtroSaludo.setAlignment (Label.CENTER);

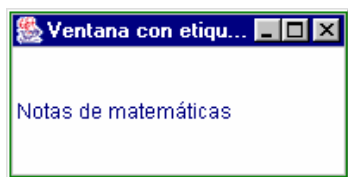
Label Cabecera = new Label ();

Cabecera.setAlignment (Label.RIGHT);

Cabecera.setText ("Ingresos totales");

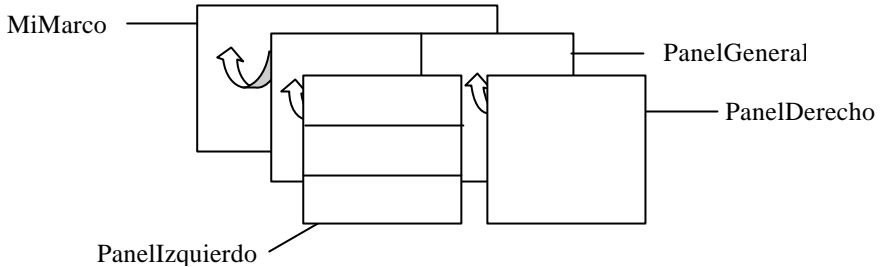
En la clase *Etiqueta1* se presenta la manera más simple de visualizar una etiqueta:

```
1 import java.awt.Frame;  
2 import java.awt.Label;  
3  
4 public class Etiqueta1 {  
5  
6     public static void main(String[] args) {  
7         Frame MiMarco = new Frame();  
8         Label Titulo = new Label("Notas de matemáticas");  
9  
10        MiMarco.add(Titulo);  
11  
12        MiMarco.setSize(200,100);  
13        MiMarco.setTitle("Ventana con etiqueta");  
14        MiMarco.setVisible(true);  
15    }  
16 }
```



El siguiente ejemplo que se presenta (*Etiqueta2*) combina el uso de diferentes “layouts” con el de etiquetas. Se instancia un *PanelGeneral* con un *GridLayout* 1 x 2 asociado (línea 7). Al *PanelGeneral* se le añaden los paneles *PanelIzquierdo* y *PanelDerecho* (líneas 11 y 12), el primero con disposición

GridLayout 3 x 1 (línea 8) y el segundo con disposición *FlowLayout* (línea 9). En las líneas 13 a 17 se definen y añaden etiquetas utilizando su constructor más completo.



```

1  import java.awt.*;
2
3  public class Etiqueta2 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel PanelGeneral = new Panel(new GridLayout(1,2));
8          Panel PanelIzquierdo = new Panel(new GridLayout(3,1));
9          Panel PanelDerecho = new Panel(new FlowLayout());
10
11         PanelGeneral.add(PanelIzquierdo);
12         PanelGeneral.add(PanelDerecho);
13         PanelIzquierdo.add(new Label("Ford",Label.CENTER));
14         PanelIzquierdo.add(new Label("Opel",Label.CENTER));
15         PanelIzquierdo.add(new Label("Audi",Label.CENTER));
16         PanelDerecho.add(new Label("Coupe"));
17         PanelDerecho.add(new Label("Cabrio"));
18
19         MiMarco.add(PanelGeneral);
20         MiMarco.setSize(250,100);
21         MiMarco.setTitle("Ventana con etiqueta");
22         MiMarco.setVisible(true);
23     }
24 }

```



En el último ejemplo de etiquetas (*Etiqueta3*) se imprime la tabla de multiplicar con cabeceras, para ello se crea un *GridLayout* de 11 x 11 (línea 7) que albergará los resultados de la tabla de multiplicar (10 x 10 resultados) más la cabecera de filas y la cabecera de columnas.

Los valores de las cabeceras de filas y columnas se crean como sendas matrices unidimensionales de etiquetas *CabeceraFila* y *CabeceraColumna* (líneas 8 y 9).

En las líneas 11 a 14 se asignan valores a los elementos del array *CabeceraFila*, y se les asocia el color rojo mediante el método *setBackground* perteneciente a la clase *Component*. En las líneas 18 a 20 se hace lo propio con el array *CabeceraColumna*.

Las etiquetas de las celdas centrales de la tabla (los resultados) se calculan y añaden al panel *Tabla* en las líneas 22 y 23, dentro del bucle que recorre cada fila (línea 18).

```
1  import java.awt.*;
2
3  public class Etiqueta3 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel Tabla = new Panel(new GridLayout(11,11));
8          Label[] CabeceraFila = new Label[11];
9          Label[] CabeceraColumna = new Label[11];
10
11         Tabla.add(new Label(""));
12         for (int i=1;i<=10;i++) {
13             CabeceraFila[i] = new Label(""+i);
14             CabeceraFila[i].setBackground(Color.red);
15             Tabla.add(CabeceraFila[i]);
16         }
17
18         for (int i=1;i<=10;i++) {
19             CabeceraColumna[i] = new Label(""+i);
20             CabeceraColumna[i].setBackground(Color.red);
21             Tabla.add(CabeceraColumna[i]);
22             for (int j=1;j<=10;j++)
23                 Tabla.add(new Label(""+i*j));
24         }
25
26         MiMarco.add(Tabla);
27         MiMarco.setSize(400,400);
28         MiMarco.setTitle("Tabla de multiplicar");
```

```
29     MiMarco.setVisible(true);
30 }
31 }
```



	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

7.3.3 Campo de texto (TextField)

Un campo de texto es un componente que permite la edición de una línea de texto. Sus constructores permiten que su tamaño tenga un número de columnas determinado y que el campo presente un texto inicial.

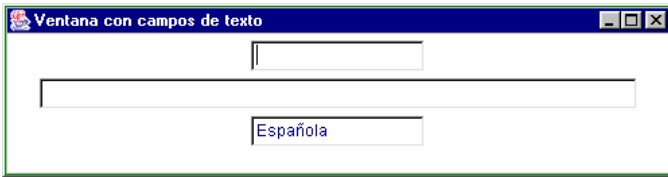
En la clase *CampoDeTexto1* se instancian tres campos de texto: *Nombre*, *Apellidos* y *Nacionalidad* (líneas 8 a 10); *Nombre* se define con un tamaño de edición de 15 columnas y sin texto inicial, *Apellidos* con 60 columnas y sin texto inicial, finalmente *Nacionalidad* con 15 columnas y texto inicial “Española”.

```
1  import java.awt.*;
2
3  public class CampoDeTexto1 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel EntradaDeDatos = new Panel(new FlowLayout());
8          TextField Nombre = new TextField(15);
9          TextField Apellidos = new TextField(60);
10         TextField Nacionalidad = new TextField("Española",15);
11
12         EntradaDeDatos.add(Nombre);
13         EntradaDeDatos.add(Apellidos);
14         EntradaDeDatos.add(Nacionalidad);
```

```

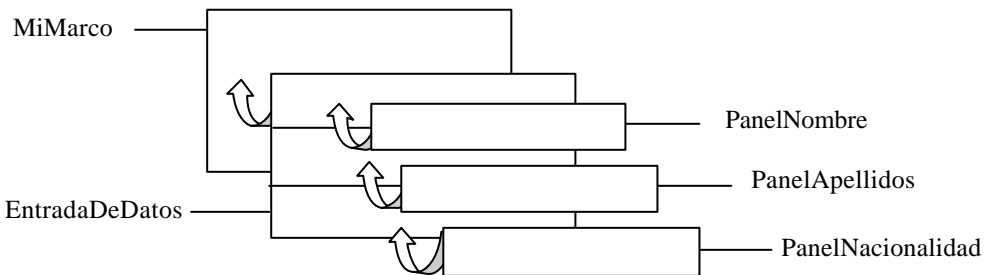
15
16     MiMarco.add(EntradaDeDatos);
17     MiMarco.setSize(500,130);
18     MiMarco.setTitle("Ventana con campos de texto");
19     MiMarco.setVisible(true);
20 }
21 }

```



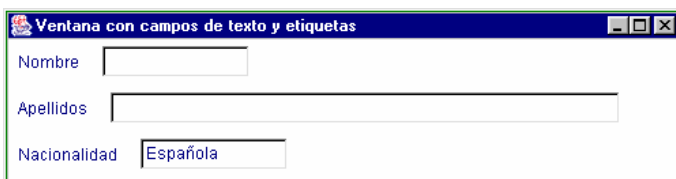
En el resultado de la clase *CampoDeTexto1* se puede observar que los campos de texto no incluyen un literal que indique lo que hay que introducir, por lo que habitualmente se utilizan etiquetas antes de los campos de texto para realizar esta función. Por otra parte, la alineación *CENTER* que por defecto presenta la disposición *FlowLayout* no siempre es adecuada. El siguiente ejemplo (*CampoDeTexto2*) resuelve ambas situaciones.

En la clase *CampoDeTexto2* se crea el panel *EntradaDeDatos* como *GridLayout* de 3 x 1, es decir, con 3 celdas verticales (línea 7). En cada una de estas celdas se añade un panel con disposición *FlowLayout* y sus componentes alineados a la izquierda (líneas 8, 9 y 10).

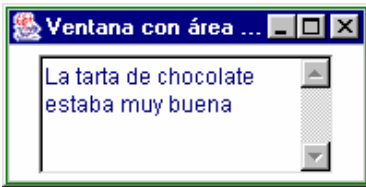


En *PanelNombre* se añaden (líneas 22 y 23) la *EtiquetaNombre* y el *CampoNombre*, definidos en las líneas 15 y 12; en *PanelApellidos* se hace lo mismo con *EtiquetaApellidos* y *CampoApellidos*; en *PanelNacionalidad* se realiza el mismo proceso. El resultado esperado se puede observar en la ventana situada tras el código.


```
1  import java.awt.*;
2
3  public class CampoDeTexto2 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel EntradaDeDatos = new Panel(new GridLayout(3,1));
8          Panel PanelNombre = new Panel(new
9                                  FlowLayout(FlowLayout.LEFT));
10         Panel PanelApellidos = new Panel(new
11                                 FlowLayout(FlowLayout.LEFT));
12         Panel PanelNacionalidad = new Panel(new
13                                 FlowLayout(FlowLayout.LEFT));
14         TextField CampoNombre = new TextField(12);
15         TextField CampoApellidos = new TextField(50);
16         TextField CampoNacionalidad = new
17                                 TextField("Española",12);
18         Label EtiquetaNombre = new Label("Nombre",Label.LEFT);
19         Label EtiquetaApellidos = new
20                                 Label("Apellidos",Label.LEFT);
21         Label EtiquetaNacionalidad = new
22                                 Label("Nacionalidad",Label.LEFT);
23
24         EntradaDeDatos.add(PanelNombre);
25         EntradaDeDatos.add(PanelApellidos);
26         EntradaDeDatos.add(PanelNacionalidad);
27         PanelNombre.add(EtiquetaNombre);
28         PanelNombre.add(CampoNombre);
29         PanelApellidos.add(EtiquetaApellidos);
30         PanelApellidos.add(CampoApellidos);
31         PanelNacionalidad.add(EtiquetaNacionalidad);
32         PanelNacionalidad.add(CampoNacionalidad);
33
34         MiMarco.add(EntradaDeDatos);
35         MiMarco.setSize(500,130);
36         MiMarco.setTitle("Ventana con campos de texto
37                             y etiquetas");
38         MiMarco.setVisible(true);
39     }
40 }
```




```
12     Comentarios.append(" de chocolate estaba buena");
13     Comentarios.insert(" muy", 28);
14
15     MiPanel.add(Comentarios);
16     MiMarco.add(MiPanel);
17     MiMarco.setSize(200,100);
18     MiMarco.setTitle("Ventana con área de texto");
19     MiMarco.setVisible(true);
20 }
21 }
```



7.3.5 Fuentes (Font)

Las fuentes no son componentes; son clases que heredan de forma directa de *Object* (la clase inicial de la jerarquía). El conocimiento de la clase *Font* es importante debido a que nos permite variar el aspecto de los textos involucrados en los componentes que estamos estudiando. Podemos cambiar el tamaño y el tipo de letra de los caracteres contenidos en una etiqueta, un campo de texto, un área de texto, etc.

Su constructor más utilizado es:

Font (String Nombre, int Estilo, int Tamaño)

El nombre, que puede ser lógico o físico, indica el tipo de letra con el que se visualizará el texto (*Serif*, *SansSerif*, *Monospaced*...). El estilo se refiere a letra itálica, negrilla, normal o alguna combinación de ellas: *Font.ITALIC*, *Font.BOLD*, *Font.PLAIN*.

La clase de ejemplo *Fuente* ilustra el modo con el que se puede utilizar el objeto *Font*. En las líneas 7 y 8 se instancian dos tipos de letras (de fuentes) a los que denominamos *UnaFuente* y *OtraFuente*, teniendo la segunda un tamaño superior a la primera (40 frente a 20).

Las líneas 14 y 17 establecen el tipo de fuente de los objetos (las etiquetas *HolaAmigo1* y *HolaAmigo2*) a los que se aplica el método *setFont*, que pertenece a la clase *Component*. Obsérvese que la mayor parte de las clases de AWT que estamos estudiando son subclases de *Component* y por lo tanto heredan su método *setFont*. Esta es la clave que nos permite aplicar fuentes a componentes. Lo mismo ocurre con el objeto *Color* y los métodos *setForeground* (líneas 15 y 18) y *setBackground*.

```
1  import java.awt.*;
2
3  public class Fuente {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Font UnaFuente = new Font("SansSerif",Font.BOLD,20);
8          Font OtraFuente = new Font("Serif",Font.ITALIC,40);
9
10         Panel Sencillo = new Panel();
11         Label HolaAmigo1 = new Label("Hola amigo");
12         Label HolaAmigo2 = new Label("Hola amigo");
13
14         HolaAmigo1.setFont(UnaFuente);
15         HolaAmigo1.setForeground(Color.red);
16
17         HolaAmigo2.setFont(OtraFuente);
18         HolaAmigo2.setForeground(Color.orange);
19
20         Sencillo.add(HolaAmigo1);
21         Sencillo.add(HolaAmigo2);
22
23         MiMarco.add(Sencillo);
24         MiMarco.setSize(500,130);
25         MiMarco.setTitle("Ventana con etiquetas y fuentes");
26         MiMarco.setVisible(true);
27     }
28 }
```

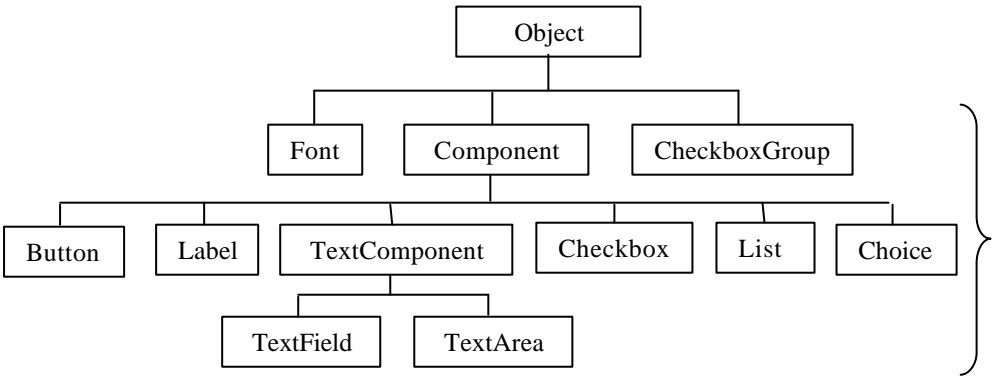


7.4 CAJAS DE VERIFICACIÓN, BOTONES DE RADIO Y LISTAS

7.4.1 Introducción

Entre los componentes habituales en el diseño de interfaces gráficas de usuario (GUI) se encuentran los botones, etiquetas, campos de texto y áreas de texto, todos ellos previamente estudiados. Ahora nos centraremos en las cajas de verificación (*Checkbox*), los botones de radio (*CheckboxGroup*), las listas (*List*) y las listas desplegables (*Choice*).

El siguiente diagrama sitúa todas estas clases en la jerarquía que proporciona Java:



7.4.2 Cajas de verificación (*Checkbox*)

Una caja de verificación es un componente básico que se puede encontrar en dos posibles estados: activado (*true*) y desactivado (*false*). Cuando se pulsa en una caja de verificación se conmuta de estado.

Los constructores más simples de la clase *Checkbox* son:

```
Checkbox()  
Checkbox(String Etiqueta)  
Checkbox(String Etiqueta, boolean Estado)
```

Entre los métodos existentes tenemos:

setLabel (String Etiqueta)
setState (boolean Estado)

De esta manera podemos definir e instanciar cajas de verificación de diversas formas:

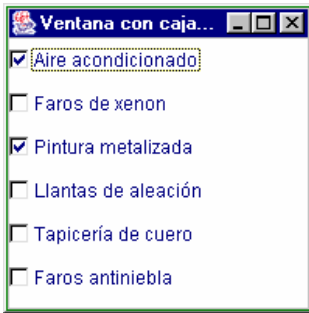
```
Checkbox MayorDeEdad = new Checkbox ("Mayor de edad", true);
```

```
Checkbox MayorDeEdad = new Checkbox ();  
MayorDeEdad.setLabel ("Mayor de edad");  
MayorDeEdad.setState ("true");
```

En la clase *CajaDeVerificación* se instancian varios componentes de tipo *Checkbox*, añadiéndolos a un panel *Sencillo* con disposición *GridLayout* 6 x 1. En la ventana obtenida se puede observar como los componentes instanciados con estado *true* aparecen inicialmente marcados.

```
1  import java.awt.*;  
2  
3  public class CajaDeVerificacion {  
4  
5      public static void main(String[] args) {  
6          Frame MiMarco = new Frame();  
7          Panel Sencillo = new Panel(new GridLayout(6,1));  
8  
9          Checkbox AireAcondicionado = new Checkbox("Aire  
10                                     acondicionado",true);  
11          Checkbox FarosXenon          = new Checkbox("Faros de  
12                                     xenon",false);  
13          Checkbox PinturaMetalizada = new Checkbox("Pintura  
14                                     metalizada",true);  
15          Checkbox LlantasAleacion    = new Checkbox("Llantas de  
16                                     aleación",false);  
17          Checkbox TapiceriaCuero     = new Checkbox("Tapicería de  
18                                     cuero",false);  
19          Checkbox FarosAntiniebla    = new Checkbox("Faros  
20                                     antiniebla",false);  
21  
22          Sencillo.add(AireAcondicionado);  
23          Sencillo.add(FarosXenon);  
24          Sencillo.add(PinturaMetalizada);  
25          Sencillo.add(LlantasAleacion);  
26          Sencillo.add(TapiceriaCuero);  
27          Sencillo.add(FarosAntiniebla);  
28  
29          MiMarco.add(Sencillo);
```

```
24     MiMarco.setSize(200,200);
25     MiMarco.setTitle("Ventana con cajas de verificación");
26     MiMarco.setVisible(true);
27 }
28 }
```



7.4.3 Botones de radio (*CheckboxGroup*)

La clase *CheckboxGroup* se utiliza para agrupar un conjunto de cajas de verificación. *CheckboxGroup* no es un componente (no deriva de la clase *Component*), es una subclase de *Object*. Cuando definimos un grupo de botones de radio utilizamos varias instancias del componente *Checkbox* asociadas a un objeto *CheckboxGroup*.

Los botones de radio presentan la característica de que la activación de un elemento implica directamente la desactivación automática de los demás botones del grupo, es decir no puede haber dos o más botones activos al mismo tiempo.

Sólo existe el constructor sin argumentos del objeto *CheckboxGroup*: *CheckboxGroup()*.

Cada caja de verificación se puede asociar a un *CheckboxGroup* de dos posibles maneras:

- Instanciando la caja de verificación con uno de los siguientes constructores:

Checkbox (String Etiqueta, boolean Estado, CheckboxGroup GrupoDeBotonesDeRadio)

Checkbox (String Etiqueta, CheckboxGroup GrupoDeBotonesDeRadio, boolean Estado)

- Utilizando el método `setCheckboxGroup` (`CheckboxGroup GrupoDeBotonesDeRadio`) sobre la caja de verificación.

El siguiente ejemplo (*BotonesDeRadio1*) instancia un grupo de botones de radio *Colores* en la línea 8. En las líneas 10 a 15 se crean seis cajas de verificación asociadas al grupo *Colores*. Estas 6 cajas de verificación se comportarán como un grupo de botones de radio (con selección excluyente).

```
1  import java.awt.*;
2
3  public class BotonesDeRadio1 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel Sencillo = new Panel(new GridLayout(6,1));
8          CheckboxGroup Colores = new CheckboxGroup();
9
10         Sencillo.add(new Checkbox("Rojo",false,Colores));
11         Sencillo.add(new Checkbox("Azul",false,Colores));
12         Sencillo.add(new Checkbox("Verde",false,Colores));
13         Sencillo.add(new Checkbox("Amarillo",false,Colores));
14         Sencillo.add(new Checkbox("Negro",false,Colores));
15         Sencillo.add(new Checkbox("Gris",false,Colores));
16
17         MiMarco.add(Sencillo);
18         MiMarco.setSize(200,200);
19         MiMarco.setTitle("Ventana con botones de radio");
20         MiMarco.setVisible(true);
21     }
22 }
```



Vamos a combinar el ejemplo anterior con el uso de fuentes (*Font*), de manera que las etiquetas de las cajas de verificación presenten un tamaño mayor y el color que representan. Los cambios más significativos respecto a nuestro ejemplo base son:

- En la línea 25 se crea una fuente *MiFuente* con tipo de letra *SansSerif*, plana (ni itálica ni negrilla) y de tamaño 25.
- En la línea 5 se define un método *EstableceVisualizacion* con argumentos: un objeto *Checkbox*, un objeto *Color* y un objeto *Font*. En la línea 8 se establece como fuente del objeto *Checkbox* el objeto *Font* que se pasa como parámetro. Análogamente, en la línea 9, se establece como color del objeto *Checkbox* el objeto *Color* que se pasa como parámetro.
- Las líneas 27 a 32 invocan al método *EstableceVisualizacion* con cada una de las cajas de verificación instanciadas en las líneas 18 a 23. En todos los casos se utiliza la misma fuente (*MiFuente*).

```
1  import java.awt.*;
2
3  public class BotonesDeRadio2 {
4
5      private static void EstableceVisualizacion(Checkbox
6          Elemento, Color ColorSeleccionado,
7          Font FuenteSeleccionada){
8          Elemento.setFont(FuenteSeleccionada);
9          Elemento.setForeground(ColorSeleccionado);
10     }
11
12
13     public static void main(String[] args) {
14         Frame MiMarco = new Frame();
15         Panel Sencillo = new Panel(new GridLayout(6,1));
16         CheckboxGroup Colores = new CheckboxGroup();
17
18         Checkbox Rojo      = new Checkbox("Rojo",false,Colores);
19         Checkbox Azul      = new Checkbox("Azul",false,Colores);
20         Checkbox Verde     = new
21             Checkbox("Verde",false,Colores);
22         Checkbox Amarillo  = new
23             Checkbox("Amarillo",false,Colores);
24         Checkbox Negro     = new
25             Checkbox("Negro",false,Colores);
26         Checkbox Gris      = new Checkbox("Gris",false,Colores);
27
28         Font MiFuente = new Font("SansSerif",Font.PLAIN,25);
29
30         EstableceVisualizacion(Rojo,Color.red,MiFuente);
31         EstableceVisualizacion(Azul,Color.blue,MiFuente);
32         EstableceVisualizacion(Verde,Color.green,MiFuente);
33         EstableceVisualizacion(Amarillo,Color.yellow,MiFuente);
34         EstableceVisualizacion(Negro,Color.black,MiFuente);
35         EstableceVisualizacion(Gris,Color.gray,MiFuente);
36     }
```

```

33
34     Sencillo.add(Rojo);
35     Sencillo.add(Azul);
36     Sencillo.add(Verde);
37     Sencillo.add(Amarillo);
38     Sencillo.add(Negro);
39     Sencillo.add(Gris);
40
41     MiMarco.add(Sencillo);
42     MiMarco.setSize(200,200);
43     MiMarco.setTitle("Ventana con botones de radio");
44     MiMarco.setVisible(true);
45 }
46 }

```



7.4.4 Lista (List)

El componente lista le ofrece al usuario la posibilidad de desplazarse por una secuencia de elementos de texto. Este componente se puede configurar de manera que se puedan realizar selecciones múltiples o bien que sólo se pueda seleccionar un único elemento de la lista.

Los constructores que admite esta clase son:

List ()

List (int NumeroDeFilas)

List (int NumeroDeFilas, boolean SeleccionMultiple)

Haciendo uso del segundo constructor podemos configurar el tamaño de la lista para que se visualicen *NumeroDeFilas* elementos de texto. Si hay más elementos en la lista se puede acceder a ellos por medio de una barra de desplazamiento vertical. Con el tercer constructor indicamos si deseamos permitir la selección múltiple de elementos.

Podemos variar en tiempo de ejecución el modo de selección de elementos, utilizando el método *setMultipleMode*(boolean *SeleccionMultiple*) de la clase *List*.

La manera de incorporar elementos de texto en una lista es mediante uno de los siguientes métodos:

```
add (String Elemento)
add (String Elemento, int Posicion)
```

Con el primer método se añade el elemento especificado al final de la lista; si se desea insertarlo en una posición determinada se utiliza el segundo método.

En el siguiente ejemplo (*Lista1*) se instancia la lista *Islas* con 5 filas y sin posibilidad de selección múltiple (línea 8). Las líneas 10 a 16 añaden elementos a la lista.

```
1  import java.awt.*;
2
3  public class Lista1 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel MiPanel = new Panel();
8          List Islas = new List(5,false);
9
10         Islas.add("Tenerife");
11         Islas.add("Lanzarote");
12         Islas.add("Gran Canaria");
13         Islas.add("Hierro");
14         Islas.add("La Gomera");
15         Islas.add("Fuerteventura");
16         Islas.add("La Palma");
17
18         MiPanel.add(Islas);
19         MiMarco.add(MiPanel);
20         MiMarco.setSize(200,200);
21         MiMarco.setTitle("Ventana con lista");
22         MiMarco.setVisible(true);
23     }
24 }
```



La clase *Lista2* parte del ejemplo anterior. En este caso se utiliza un constructor más sencillo (línea 8) y se complementa haciendo uso del método *setMultipleMode* (línea 9), con el que se indica que se permite selección múltiple (necesario, por ejemplo si existe la posibilidad de paquetes turísticos combinados entre varias islas).

Las líneas 19 y 20 utilizan el método *select*, que permite seleccionar por programa el elemento indicado por la posición que se le pasa como parámetro. En este caso se han seleccionado “Gran Canaria” y “Lanzarote”. Obsérvese que la posición de los elementos se numera a partir de cero.

```
1  import java.awt.*;
2
3  public class Lista2 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel MiPanel = new Panel();
8          List Islas = new List(3);
9          Islas.setMultipleMode(true);
10
11         Islas.add("Fuerteventura");
12         Islas.add("La Gomera");
13         Islas.add("Gran Canaria");
14         Islas.add("Hierro");
15         Islas.add("Lanzarote");
16         Islas.add("Tenerife");
17         Islas.add("La Palma",5);
18
19         Islas.select(2);
20         Islas.select(4);
21
22         MiPanel.add(Islas);
23         MiMarco.add(MiPanel);
24         MiMarco.setSize(200,200);
25         MiMarco.setTitle("Ventana con lista de selección
                               múltiple");
26         MiMarco.setVisible(true);
27     }
28 }
```



7.4.5 Lista desplegable (Choice)

La clase *Choice* presenta un menú desplegable de posibilidades. La posibilidad seleccionada aparece como título del menú. No existe la posibilidad de realizar selecciones múltiples.

Esta clase sólo admite el constructor sin parámetros: *Choice()* y sus elementos se añaden mediante el método *add(String Elemento)*.

El ejemplo *ListaDesplegable1* ilustra la manera de hacer un uso sencillo del componente *Choice*: en la línea 8 se instancia la lista desplegable *Ciudades* y en las líneas 10 a 15 se añaden los elementos deseados.

```
1  import java.awt.*;
2
3  public class ListaDesplegable1 {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7          Panel MiPanel = new Panel();
8          Choice Ciudades = new Choice();
9
10         Ciudades.add("Alicante");
11         Ciudades.add("Avila");
12         Ciudades.add("Granada");
13         Ciudades.add("Segovia");
14         Ciudades.add("Sevilla");
15         Ciudades.add("Toledo");
16
17         MiPanel.add(Ciudades);
18         MiMarco.add(MiPanel);
19         MiMarco.setSize(200,200);
20         MiMarco.setTitle("Ventana con lista desplegable");
21         MiMarco.setVisible(true);
22     }
23 }
```



Si en el ejemplo anterior introducimos entre las líneas 15 y 17 las siguientes instrucciones:

```
Ciudades.insert("Madrid",3); // inserta "Madrid" en la cuarta posición de la lista
                          // desplegable "Ciudades"
Ciudades.select("Segovia"); // selecciona el elemento de texto "Segovia"
```

Obtendremos el resultado:

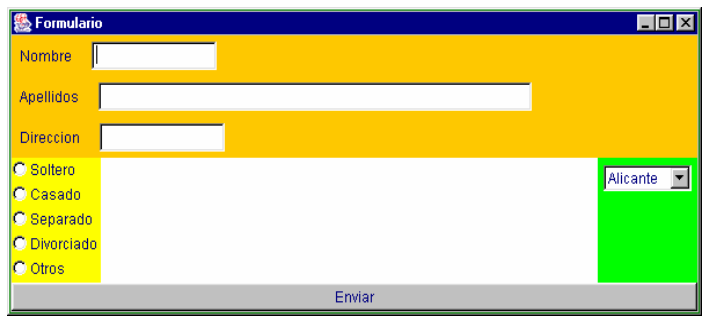


7.5 DISEÑO DE FORMULARIOS

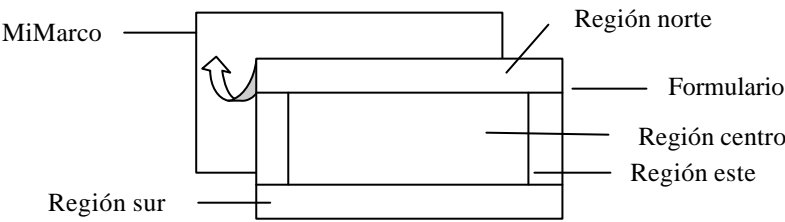
7.5.1 Diseño del interfaz gráfico deseado

Supongamos que deseamos crear una pantalla de entrada de datos que nos permita recibir información personal de usuarios, por ejemplo contribuyentes fiscales residentes en España.

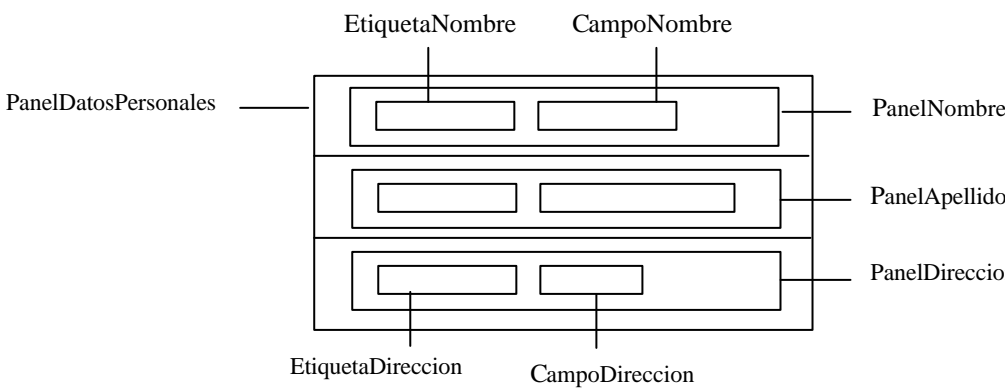
En la parte superior de la ventana deseamos preguntar el nombre y los apellidos, así como la nacionalidad, que por defecto se rellenará como española, puesto que sabemos que la gran mayoría de los contribuyentes poseen dicha nacionalidad. En la zona izquierda de la ventana se consulta el estado civil y en la derecha la ciudad de residencia; la zona central se deja libre para incluir el logotipo del ministerio de hacienda, de lo que se encargará un equipo de diseño gráfico, y la inferior contendrá el botón de enviar/grabar datos. El resultado esperado tendrá un aspecto similar a la ventana siguiente:



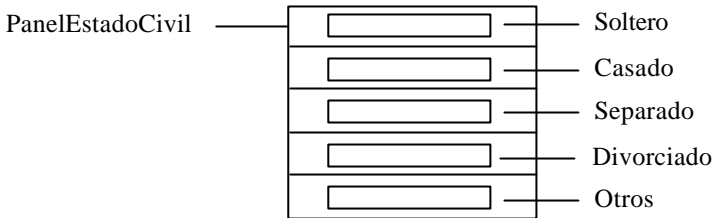
Antes de la programación deberíamos diseñar el formulario en base a contenedores (paneles) y disposiciones (*layouts*). En primer lugar necesitamos una ventana, que conseguiremos mediante un objeto *Frame* al que podemos denominar, igual que en los ejemplos anteriores, *MiMarco*. A *MiMarco* le añadiremos un panel “Formulario” con disposición *BorderLayout*, que encaja perfectamente con el diseño pedido.



El grupo de datos nombre, apellidos y nacionalidad requieren, cada uno, de una etiqueta y un campo de texto dispuestos en secuencia horizontal (*FlowLayout*) y alineados a la izquierda. A su vez estos tres grupos de componentes se deben situar en secuencia vertical, para lo que resulta adecuado un *GridLayout* 3 x 1.



Los botones de radio que sirven para seleccionar el estado civil se pueden añadir a un único panel “PanelEstadoCivil” asociado a un *GridLayout* 5 x 1:



El resto de los componentes no requiere de una disposición especial, pudiendo añadirse directamente a sus regiones correspondientes en el panel *Formulario*.

7.5.2 Implementación en una sola clase

Si pensamos que los elementos de nuestro formulario no se van a poder reutilizar en otras ventanas de GUI podemos implementar todo el diseño en una sola clase, el siguiente ejemplo (*Formulario1*) codifica el diseño realizado:

En primer lugar se instancian el marco y los paneles que hemos previsto (líneas 5 a 10), como variables globales a la clase, privadas a la misma y estáticas para poder ser usadas desde el método estático *main*.

En el método principal (*main*), en primer lugar se hacen las llamadas a los métodos privados *PreparaDatosPersonales*, *PreparaEstadoCivil* y *PreparaProvincia* (líneas 66 a 68), que añaden los componentes necesarios a los paneles *PanelDatosPersonales*, *PanelEstadoCivil* y *PanelProvincia*. Estos métodos podrían haber admitido un panel como parámetro, evitándose las variables globales y estáticas, que en este caso se habrían declarado en el propio método *main*.

En las líneas 70 a 72 se asignan los colores deseados a cada uno de los paneles principales. En las líneas 74 a 77 se añaden los paneles principales a las regiones deseadas en el panel *Formulario*. El código de cada uno de los métodos privados se obtiene de manera directa sin más que seguir los gráficos diseñados en el apartado anterior.

```
1 import java.awt.*;
2
```



```
3 public class Formulario1 {
4
5     private static Frame MiMarco = new Frame();
6     private static Panel Formulario = new Panel(new
7         BorderLayout());
8     private static Panel PanelDatosPersonales =
9         new Panel(new
10             GridLayout(3,1));
11     private static Panel PanelEstadoCivil = new Panel(new
12         GridLayout(5,1));
13     private static Panel PanelProvincia = new Panel();
14
15     private static void PreparaDatosPersonales(){
16         Panel PanelNombre = new Panel(new
17             FlowLayout(FlowLayout.LEFT));
18         Panel PanelApellidos = new Panel(new
19             FlowLayout(FlowLayout.LEFT));
20         Panel PanelDireccion = new Panel(new
21             FlowLayout(FlowLayout.LEFT));
22
23         TextField CampoNombre = new TextField(12);
24         TextField CampoApellidos = new TextField(50);
25         TextField CampoDireccion = new TextField(12);
26         Label EtiquetaNombre = new Label("Nombre",Label.LEFT);
27         Label EtiquetaApellidos = new
28             Label("Apellidos",Label.LEFT);
29         Label EtiquetaDireccion = new
30             Label("Direccion",Label.LEFT);
31
32         PanelDatosPersonales.add(PanelNombre);
33         PanelDatosPersonales.add(PanelApellidos);
34         PanelDatosPersonales.add(PanelDireccion);
35         PanelNombre.add(EtiquetaNombre);
36         PanelNombre.add(CampoNombre);
37         PanelApellidos.add(EtiquetaApellidos);
38         PanelApellidos.add(CampoApellidos);
39         PanelDireccion.add(EtiquetaDireccion);
40         PanelDireccion.add(CampoDireccion);
41     }
42
43     private static void PreparaEstadoCivil() {
44         CheckboxGroup EstadoCivil = new CheckboxGroup();
45
46         Checkbox Soltero = new
47             Checkbox("Soltero",false,EstadoCivil);
48         Checkbox Casado = new
49             Checkbox("Casado",false,EstadoCivil);
50         Checkbox Separado = new
```

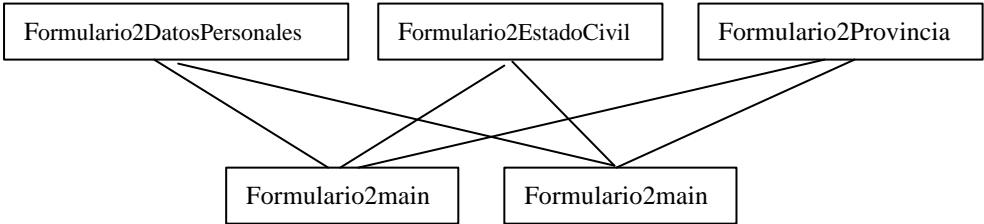
```

                                Checkbox("Separado", false, EstadoCivil);
41     Checkbox Divorciado = new
                                Checkbox("Divorciado", false, EstadoCivil);
42     Checkbox Otros      = new
                                Checkbox("Otros", false, EstadoCivil);
43
44     PanelEstadoCivil.add(Soltero);
45     PanelEstadoCivil.add(Casado);
46     PanelEstadoCivil.add(Separado);
47     PanelEstadoCivil.add(Divorciado);
48     PanelEstadoCivil.add(Otros);
49 }
50
51 private static void PreparaProvincia() {
52     Choice Ciudades = new Choice();
53
54     Ciudades.add("Alicante");
55     Ciudades.add("Avila");
56     Ciudades.add("Granada");
57     Ciudades.add("Madrid");
58     Ciudades.add("Segovia");
59     Ciudades.add("Sevilla");
60     Ciudades.add("Toledo");
61
62     PanelProvincia.add(Ciudades);
63 }
64
65 public static void main(String[] args) {
66     PreparaDatosPersonales();
67     PreparaEstadoCivil();
68     PreparaProvincia();
69
70     PanelDatosPersonales.setBackground(Color.orange);
71     PanelEstadoCivil.setBackground(Color.yellow);
72     PanelProvincia.setBackground(Color.green);
73
74     Formulario.add(PanelDatosPersonales,
                    BorderLayout.NORTH);
75     Formulario.add(PanelEstadoCivil, BorderLayout.WEST);
76     Formulario.add(PanelProvincia, BorderLayout.EAST);
77     Formulario.add(new
                    Button("Enviar"), BorderLayout.SOUTH);
78
79     MiMarco.add(Formulario);
80     MiMarco.setSize(600, 250);
81     MiMarco.setTitle("Formulario");
82     MiMarco.setVisible(true);
83 } }
```

7.5.3 Implementación en varias clases

El formulario diseñado contiene secciones que posiblemente nos interese reutilizar en otras ventanas de interfaz gráfico, por ejemplo, el panel que permite introducir el nombre, apellidos y nacionalidad con toda probabilidad podrá ser utilizado de nuevo en la misma aplicación o bien en otras diferentes.

Para facilitar la reutilización al máximo, crearemos una clase por cada uno de los paneles principales y después crearemos una o varias clases que hagan uso de las anteriores. El esquema que seguiremos es el siguiente:



En primer lugar presentamos la clase *Formulario2DatosPersonales*, en donde creamos la propiedad privada *PanelDatosPersonales* (línea 5). Esta propiedad es accesible a través del método público *DamePanel()* situado en la línea 18.

La clase tiene un constructor con argumento de tipo *Color* (línea 7), lo que nos permite asignar el color seleccionado al panel *PanelDatosPersonales* (línea 11). El segundo constructor de la clase (línea 14) no presenta parámetros; internamente llama al primero con el parámetro de color blanco.

El contenido del constructor se obtiene directamente del método *PreparaDatosPersonales()*.

```
1  import java.awt.*;
2
3  public class Formulario2DatosPersonales {
4
5      private Panel PanelDatosPersonales = new Panel(new
6                                          GridLayout(3,1));
7      Formulario2DatosPersonales(Color ColorDelPanel){
8
9          // *** Codigo del metodo PreparaDatosPersonales() ***
10
11          PanelDatosPersonales.setBackground(ColorDelPanel);
```

```
12  }
13
14  Formulario2DatosPersonales(){
15      this(Color.white);
16  }
17
18  Panel DamePanel() {
19      return PanelDatosPersonales;
20  }
21
22 }
```

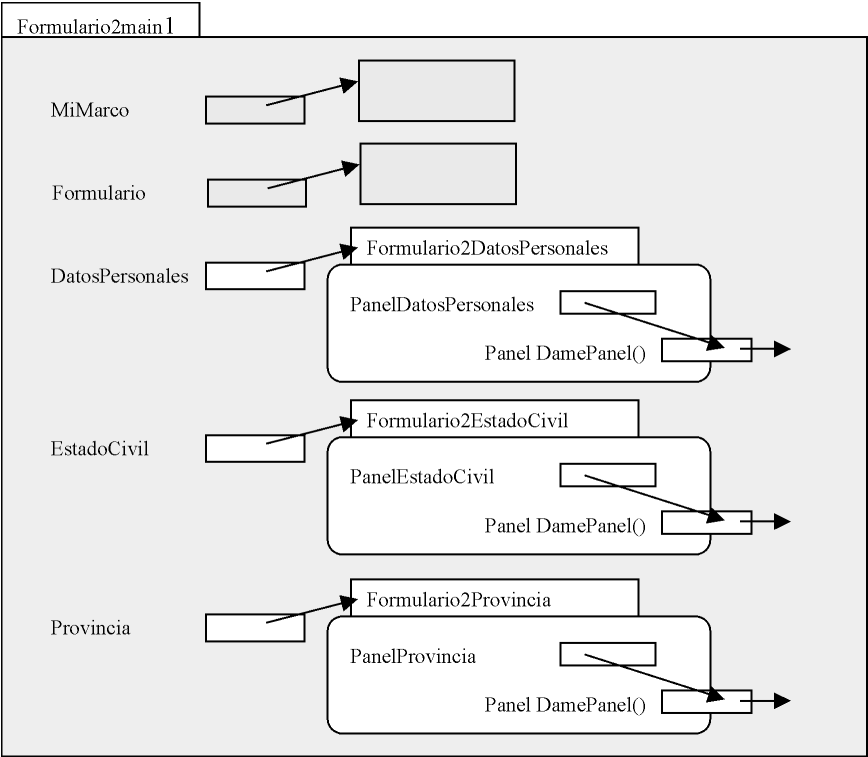
La clase *Formulario2EstadoCivil* se diseña con los mismos principios que *Formulario2DatosPersonales*. A continuación se muestra su código abreviado:

```
1  import java.awt.*;
2
3  public class Formulario2EstadoCivil {
4
5      private Panel PanelEstadoCivil = new Panel(new
6                                          GridLayout(5,1));
7
8      Formulario2EstadoCivil(Color ColorDelPanel) {
9
10         // ***Codigo del metodo PreparaEstadoCivil() ***
11
12         PanelEstadoCivil.setBackground(ColorDelPanel);
13     }
14
15     Formulario2EstadoCivil(){
16         this(Color.white);
17     }
18
19     Panel DamePanel() {
20         return PanelEstadoCivil;
21     }
22 }
```

Finalmente se presenta el código resumido de la clase *Formulario2Provincia*:

```
1  import java.awt.*;
2
3  public class Formulario2Provincia {
4
5      private Panel PanelProvincia = new Panel();
```

```
6
7  Formulario2Provincia(Color ColorDelPanel){
8
9      // *** Codigo del metodo PreparaProvincia() ***
10
11      PanelProvincia.setBackground(ColorDelPanel);
12  }
13
14  Formulario2Provincia() {
15      this(Color.white);
16  }
17
18  Panel DamePanel() {
19      return PanelProvincia;
20  }
21
22 }
```



La clase *Formulario2main1* presenta exactamente el mismo resultado que la clase *Formulario1*, por ello no representamos de nuevo la ventana de resultado.

En primer lugar se crean las propiedades de clase *MiMarco* y *Formulario* (también se podrían haber creado como variables de instancia dentro del método *main*), después, en el método *main*, se crean sendas instancias de los objetos *Formulario2DatosPersonales*, *Formulario2EstadoCivil* y *Formulario2Provincia* (líneas 9 a 13), cada uno con el color deseado.

En este momento se encuentran preparados (como estructuras de datos) los paneles *PanelDatosPersonales*, *PanelEstadoCivil* y *PanelProvincia*, cada uno como propiedad privada de su objeto correspondiente (*DatosPersonales*, *EstadoCivil* y *Provincia*).

En las líneas 15 a 17 se obtienen los paneles *PanelDatosPersonales*, *PanelEstadoCivil* y *PanelProvincia* invocando al método *DamePanel()* en las instancias *DatosPersonales*, *EstadoCivil* y *Provincia*. En estas mismas líneas se añaden los paneles obtenidos a las regiones deseadas del panel *Formulario*.

```
1  import java.awt.*;
2
3  public class Formulario2main1 {
4
5      private static Frame MiMarco = new Frame();
6      private static Panel Formulario = new Panel(new
                                   BorderLayout());
7
8      public static void main(String[] args) {
9          Formulario2DatosPersonales DatosPersonales =
10              new Formulario2DatosPersonales(Color.orange);
11          Formulario2EstadoCivil EstadoCivil =
12              new Formulario2EstadoCivil(Color.yellow);
13          Formulario2Provincia Provincia = new
                                   Formulario2Provincia(Color.green);
14
15          Formulario.add(DatosPersonales.DamePanel(),
                                   BorderLayout.NORTH);
16          Formulario.add(EstadoCivil.DamePanel(),
                                   BorderLayout.WEST);
17          Formulario.add(Provincia.DamePanel(),
                                   BorderLayout.EAST);
18          Formulario.add(new Button("Enviar"),
                                   BorderLayout.SOUTH);
19
20          MiMarco.add(Formulario);
21          MiMarco.setSize(600,250);
22          MiMarco.setTitle("Formulario");
```

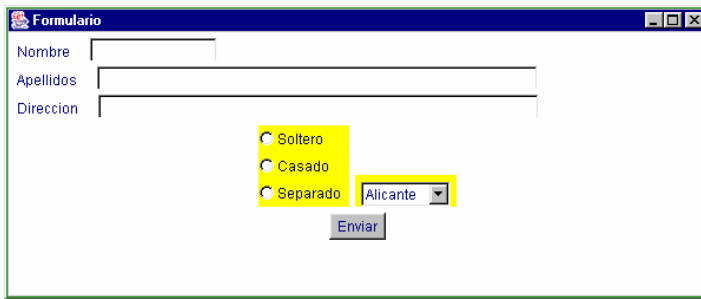
```

23     MiMarco.setVisible(true);
24 }
25 }

```

La clase *Formulario2main2* hace uso de los objetos *Formulario2DatosPersonales*, *Formulario2EstadoCivil* y *Formulario2Provincia* para crear un interfaz de usuario con la misma funcionalidad que la del ejemplo anterior, pero con una disposición diferente de los componentes.

La ventana de resultado se presenta a continuación y la naturaleza del código es muy similar a la del ejemplo anterior. Al reutilizar las clases básicas tenemos la posibilidad de cambiar la apariencia final de la ventana de interfaz de usuario.



```

1  import java.awt.*;
2
3  public class Formulario2main2 {
4
5      private static Frame MiMarco = new Frame();
6      private static Panel Formulario = new Panel(new
                                                GridLayout(3,1));
7
8      public static void main(String[] args) {
9
10         Panel EstadoCivil_Provincia =
11             new Panel(new FlowLayout(FlowLayout.CENTER));
12         Panel BotonEnviar = new Panel();
13
14         Formulario2DatosPersonales DatosPersonales =
15             new Formulario2DatosPersonales();
16         Formulario2EstadoCivil EstadoCivil =
17             new Formulario2EstadoCivil(Color.yellow);
18         Formulario2Provincia Provincia = new
19             Formulario2Provincia(Color.yellow);

```

```
20     EstadoCivil_Provincia.add(EstadoCivil.DamePanel());
21     EstadoCivil_Provincia.add(Provincia.DamePanel());
22     BotonEnviar.add(new Button("Enviar"));
23
24     Formulario.add(DatosPersonales.DamePanel());
25     Formulario.add(EstadoCivil_Provincia);
26     Formulario.add(BotonEnviar);
27
28     MiMarco.add(Formulario);
29     MiMarco.setSize(600,250);
30     MiMarco.setTitle("Formulario");
31     MiMarco.setVisible(true);
32 }
33 }
```

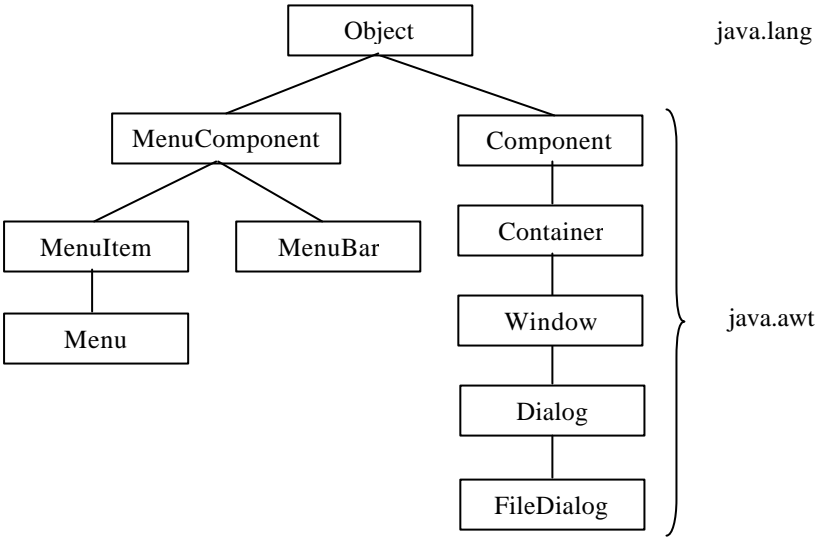
Este mismo ejercicio se podría haber resuelto con un enfoque más elegante desde el punto de vista de programación orientada a objetos: podríamos haber definido las clases *Formulario2DatosPersonales*, *Formulario2EstadoCivil* y *Formulario2Provincia* como subclases de *Panel*. En ese caso no necesitaríamos crear instancias del objeto *Panel* dentro de cada una de las tres clases, ni tampoco haría falta definir y utilizar el método *DamePanel()*.

7.6 DIÁLOGOS Y MENÚS

7.6.1 Introducción

Los siguientes componentes de AWT que vamos a estudiar son los diálogos, con los que podemos crear ventanas adicionales a la principal, especializadas en diferentes tipos de interacción con el usuario (por ejemplo selección de un fichero). También nos centraremos en la manera de crear menús.

Los diálogos los basaremos en las clases *Dialog* y *FileDialog*, mientras que los menús los crearemos con los objetos *MenuBar* y *Menu*. El siguiente diagrama sitúa todas estas clases en la jerarquía que proporciona Java:



7.6.2 *Diálogo (Dialog)*

Un diálogo es una ventana que se usa para recoger información del usuario; su disposición por defecto es *BorderLayout*. Para poder crear un diálogo hay que indicar quién es su propietario (un marco o bien otro diálogo).

Las ventanas de diálogo pueden ser modales o no modales (opción por defecto). Las ventanas de diálogo modales se caracterizan porque realizan un bloqueo de las demás ventanas de la aplicación (salvo las que hayan podido ser creadas por el propio diálogo).

Existen 5 constructores de la clase *Dialog*, los más utilizados son:

- Dialog (Frame Propietario, String Titulo, boolean Modal)*
- Dialog (Dialog Propietario, String Titulo, boolean Modal)*
- Dialog (Frame Propietario)*
- Dialog (Dialog Propietario)*

Entre los métodos existentes tenemos:

- setTitle (String Titulo)*
- setModal (boolean Modal)*
- hide() // oculta el diálogo*
- show() // muestra el diálogo*
- setResizable(boolean CambioTamano) // permite o no el cambio de tamaño de la ventana por parte del usuario*

El primer ejemplo del componente diálogo (*Dialogo1*), en la línea 9 instancia un diálogo *Dialogo* cuyo propietario es el marco *MiMarco*, el título “Ventana de diálogo” y su naturaleza no modal (parámetro a *false*).

La línea 14 invoca al método *show* del objeto *Dialogo* para conseguir que la ventana se haga visible. El diálogo que se obtiene es una ventana vacía y sin dimensiones.

```
1  import java.awt.*;
2
3  public class Dialogo1 {
4
5      public static void main(String[] args) {
6          final boolean NO_MODAL = false;
7          Frame MiMarco = new Frame();
8
9          Dialog Dialogo = new Dialog(MiMarco,"Ventana de
                                diálogo",NO_MODAL);
10
11         MiMarco.setSize(200,100);
12         MiMarco.setTitle("Ventana con diálogo");
13         MiMarco.setVisible(true);
14         Dialogo.show();
15     }
16 }
```



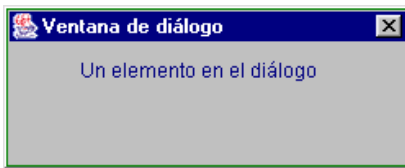
En la clase *Dialogo2* se crea un diálogo modal de propietario *MiMarco* (línea 11). Con el diálogo (*Dialogo*) instanciado trabajamos de manera análoga a como lo hemos venido haciendo con los marcos. En el ejemplo se le añade un panel (línea 12), se le asigna un tamaño (línea 13) y también una posición (línea 14). Al igual que en el ejemplo anterior, debemos mostrar el componente (línea 19).

```
1  import java.awt.*;
2
3  public class Dialogo2 {
4
5      public static void main(String[] args) {
6          final boolean MODAL = true;
7          Frame MiMarco = new Frame();
8          Panel MiPanel = new Panel();
9          MiPanel.add(new Label("Un elemento en el diálogo"));
10
11         Dialog Dialogo = new Dialog(MiMarco,"Ventana de
```

```

12         dialogo.add(MiPanel);
13         dialogo.setSize(250,100);
14         dialogo.setLocation(new Point(50,80));
15
16         MiMarco.setSize(200,100);
17         MiMarco.setTitle("Ventana con diálogo");
18         MiMarco.setVisible(true);
19         dialogo.show();
20     }
21 }

```



7.6.3 Diálogo de carga / almacenamiento de ficheros (FileDialog)

FileDialog es una subclase de *Dialog*. Como clase especializada de *Dialog*, *FileDialog* gestiona la selección de un fichero entre los sistemas de ficheros accesibles. Los diálogos de fichero son modales y sus propietarios deben ser marcos.

Los constructores de la clase son:

FileDialog (Frame Propietario)

FileDialog (Frame Propietario, String Titulo)

FileDialog (Frame Propietario, String Titulo, int ModoCargaOGrabacion)

Entre los métodos más utilizados de esta clase están:

getMode() y *setMode(int ModoCargaOGrabacion)*

getFile() y *setFile(String NombreFichero)* // para seleccionar por programa el
// fichero (o consultar cuál

// es el fichero que ha sido seleccionado)

getDirectory y *setDirectory(String NombreDirectorio)* // para seleccionar el
// directorio inicial que

// utilizará el diálogo (o consultarlo)

getFilenameFilter() y *setFilenameFilter(FilenameFilter Filtro)* // para mostrar
// únicamente los ficheros que pasan el filtro

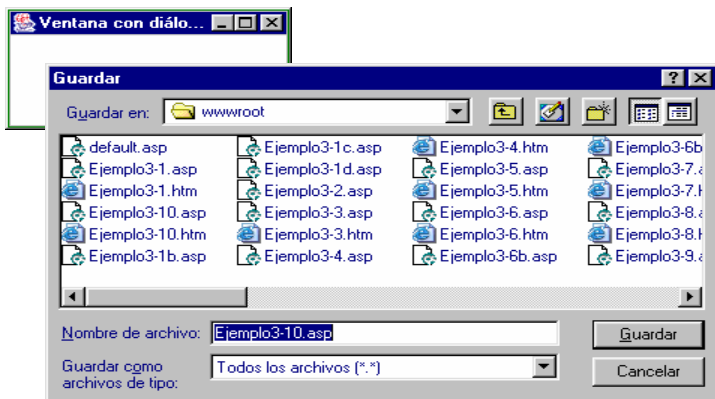
En el siguiente ejemplo (*DialogoFichero*) se crea una instancia *Grabar* de la clase *FileDialog* (línea 8). El diálogo tiene como propietario el marco *MiMarco*, como título el texto “Guardar” y como modo *SAVE*. La línea 9 (comentada) ilustra la forma de crear un diálogo de fichero en modo *LOAD*.

En la línea 15 se invoca al método *show*, perteneciente a la clase *Dialog* (superclase de *FileDialog*).

```

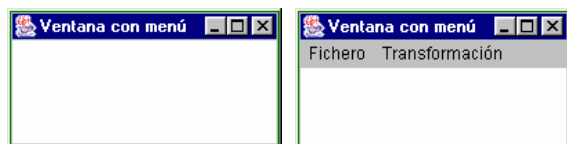
1  import java.awt.*;
2
3  public class DialogoFichero {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7
8          FileDialog Grabar = new
9              FileDialog(MiMarco, "Guardar", FileDialog.SAVE);
10         // FileDialog Cargar =
11             new FileDialog(MiMarco, "Cargar",
12                 FileDialog.LOAD);
13
14         MiMarco.setSize(200,100);
15         MiMarco.setTitle("Ventana con diálogo de carga
16                             de fichero");
17         MiMarco.setVisible(true);
18         Grabar.show();
19     }
20 }

```



7.6.4 Menús (*Menu* y *MenuBar*)

La clase *MenuBar* se utiliza para dotar de una barra principal de menú a un marco. A continuación se presenta un marco sin barra de menú y otro con una barra de menú compuesto de dos menús.



MenuBar solo tiene el constructor sin argumentos. En la línea 8 de la clase *Menus* se instancia una barra de menú con identificador *MenuPrincipal*.

Para añadir una barra de menú a un marco se emplea el método *setMenuBar* (no *add*), tal y como aparece en la línea 24 del ejemplo; *setMenuBar* es un método de la clase *Frame*.

Una vez que disponemos de un objeto barra de menú, podemos añadirle menús (con el método *add*), tal y como se realiza en las líneas 21 y 22 del ejemplo. Veamos primero qué es y como se emplea un menú: un menú es un componente desplegable que parte de una barra de menús. Su representación gráfica se puede ver en las ventanas situadas al final del código de la clase *Menus*.

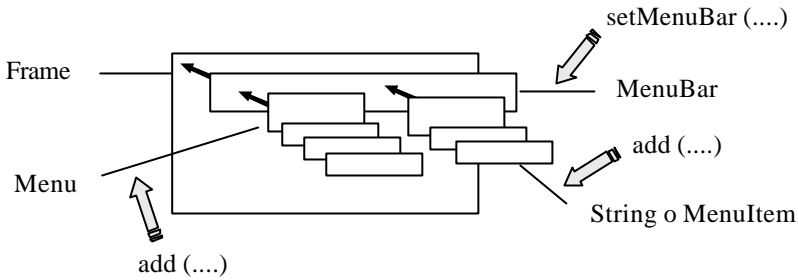
El constructor más utilizado de la clase menú es:

Menu (*String* *Etiqueta*)

El constructor definido crea un menú vacío con la etiqueta de título que se le pasa como parámetro; en nuestro ejemplo tenemos un menú *Fichero* con etiqueta “Fichero” (línea 9) y un menú *Transformaciones* con etiqueta “Transformación” (línea 10). La etiqueta representa el texto que se visualiza en la barra de menús.

En las líneas 12 a 19 del ejemplo se añaden (usando el método *add*) las diferentes opciones que el usuario puede seleccionar en los menús.

El siguiente esquema muestra la manera en la que se disponen las clases involucradas, así como los métodos necesarios para asociarlas entre sí.



```

1  import java.awt.*;
2
3  public class Menus {
4
5      public static void main(String[] args) {
6          Frame MiMarco = new Frame();
7
8          MenuBar MenuPrincipal = new MenuBar();
9          Menu Fichero = new Menu("Fichero");
10         Menu Transformaciones = new Menu("Transformación");
11
12         Fichero.add("Abrir");
13         Fichero.add("Cerrar");
14         Fichero.add("Imprimir");
15         Fichero.add("Guardar");
16
17         Transformaciones.add("Rotación");
18         Transformaciones.add("Traslación");
19         Transformaciones.add("Cambio de escala");
20
21         MenuPrincipal.add(Fichero);
22         MenuPrincipal.add(Transformaciones);
23
24         MiMarco.setMenuBar(MenuPrincipal);
25         MiMarco.setSize(200,100);
26         MiMarco.setTitle("Ventana con menú");
27         MiMarco.setVisible(true);
28     }
29 }

```

